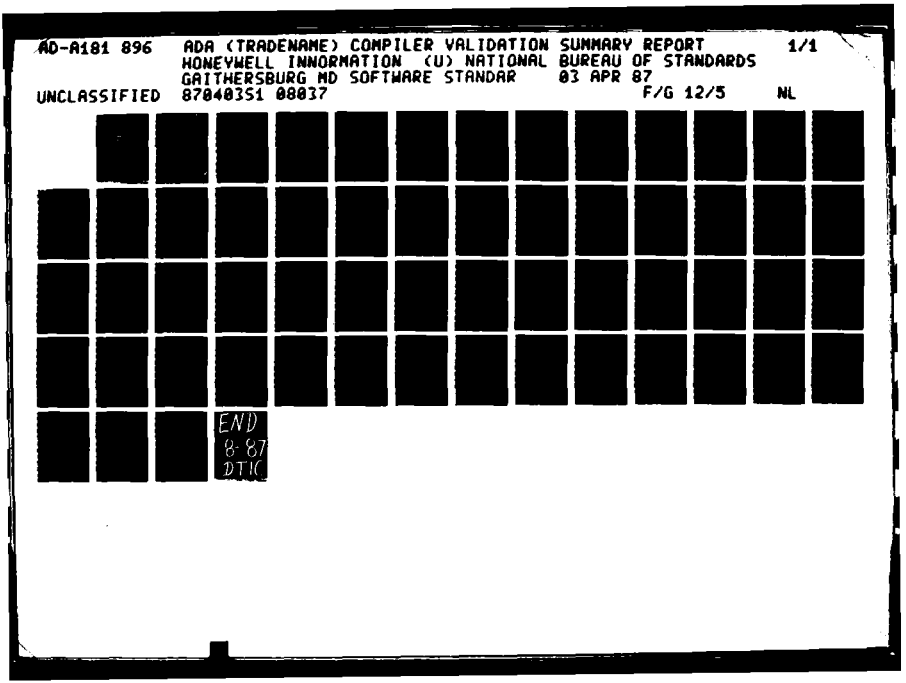

AD-A181 896 ADA (TRADENAME) COMPILER VALIDATION SUMMARY REPORT 1/1
HONEYWELL INNORMATION (U) NATIONAL BUREAU OF STANDARDS
GAITHERSBURG MD SOFTWARE STANDAR 03 APR 87
UNCLASSIFIED 87040351 08037 F/G 12/5 NL
END
8-87
DTIC

MICROCOPY RESOLUTION TEST CHART

DTIC FILE COPY

REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS BEFORE COMPLETEING FORM

1. REPORT NUMBER | 2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) Ada Compiler Validation Summary Report: Honeywell Information Systems, GCOS-8, V2.0	5. TYPE OF REPORT & PERIOD COVERED 3 Apr 1987 to 3 Apr 1988 6. PERFORMING ORG. REPORT NUMBER 870403S1.08037
7. AUTHOR(s) Software Standards Validation Group Institute for Computer Sciences and Technology National Bureau of Standards	8. CONTRACT OR GRANT NUMBER(s)
9. PERFORMING ORGANIZATION AND ADDRESS Software Standards Validation Group, National Bureau of Standards, Bldg.225, Rm A266 Gaithersburg, MD 20899	10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS Ada Joint Program Office United States Department of Defense Washington, DC 20301-3081ASD/SIOL	12. REPORT DATE 3 Apr 1987 13. NUMBER OF PAGES 53
14. MONITORING AGENCY NAME & ADDRESS(If different from Controlling Office) National Bureau of Standards	15. SECURITY CLASS (of this report) UNCLASSIFIED 15a. DECLASSIFICATION/DOWNGRADING SCHEDULE N/A

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20. If different from Report)

UNCLASSIFIED

18. SUPPLEMENTARY NOTES

19. KEYWORDS (Continue on reverse side if necessary and identify by block number)

Ada Programming language, Ada Compiler Validation Summary Report, Ada Compiler Validation Capability, ACVC, Validation Testing, Ada Validation Office, AVO, Ada Validation Facility, AVF, ANSI/MIL-STD-1815A, Ada Joint Program Office, AJPO

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

See Attached.

DTIC ELECTE JUL 06 1987 S D A

AD-A181 896

DD FORM 1473 1 JAN 73 EDITION OF 1 NOV 65 IS OBSOLETE
S/N 0102-LF-014-6601

EXECUTIVE SUMMARY

This Validation Summary Report summarizes the results and conclusions of validation testing performed on the GCOS-8 V2.0 using Version 1.8 of the *Ada Compiler Validation Capability (ACVC).

The validation process includes submitting a suite of standardized tests (the ACVC) as inputs to an Ada compiler and evaluating the results. The purpose is to ensure conformance of the computer to ANSI/MIL-STD-1815A FIPS PUB 119 Ada by testing that it properly implements legal language constructs and that it identifies and rejects illegal language constructs. The testing also identifies behavior that is implementation dependent but permitted by ANSI/MIL-STD-1815A FIPS PUB 119. Six classes of tests are used. These tests are designed to perform checks at compile time, at link time, or during execution.

On-site testing was performed 30 March 1987 through 3 April 1987 in Phoenix, AZ under the auspices of the Software Standards Validation Group, according to Ada Validation Organization policies and procedures. The GCOS-8 V2.0 was hosted on DPS-90 operating under SR3000 V.30002 and SR2500 V.25003

The results of validation are summarized in the following table:

RESULT	TEST CLASS						TOTAL
	A	B	C	D	E	L	
Passed	69	867	1190	17	13	46	2202
Failed	0	0	0	0	0	0	0
Inapplicable	0	0	178	0	0	0	178
Withdrawn	0	7	12	0	0	0	19
TOTAL	69	874	1380	17	13	46	2399

*Ada is a registered trademark of the United States Government (Ada Joint Program Office).

8704035 1 ,08037

FSV87VSRHIS518A

Ada* COMPILER
VALIDATION SUMMARY REPORT:
Honeywell Information Systems
GCOS-8 V2.0

Completion of On-Site Validation:
3 April 1987

Prepared By:
Software Standards Validation Group
Institute for Computer Sciences and Technology
National Bureau of Standards
Building 225, Room A266
Gaithersburg, MD 20899

Prepared For:
Ada Joint Program Office
United States Department of Defense
Washington, D.C.

---

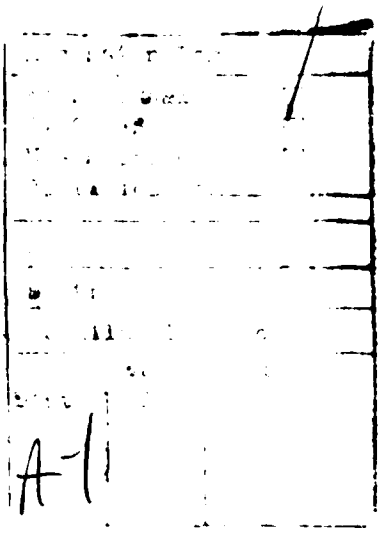
*Ada is a registered trademark of the United States Government (Ada Joint Program Office).

Ada Compiler Validation Summary Report:

Compiler Name: GCOS-8 V2.0

Host Computer:	Target Computer:
DPS 90	DPS 90
under	under
SR2500 V.25003	SR2500 V.25003
and	and
SR3000 V.30002	SR3000 V.30002

Testing Completed on 3 April 1987 Using ACVC 1.8.



EXECUTIVE SUMMARY

This Validation Summary Report summarizes the results and conclusions of validation testing performed on the GCOS-8 V2.0 using Version 1.8 of the *Ada Compiler Validation Capability (ACVC).

The validation process includes submitting a suite of standardized tests (the ACVC) as inputs to an Ada compiler and evaluating the results. The purpose is to ensure conformance of the computer to ANSI/MIL-STD-1815A FIPS PUB 119 Ada by testing that it properly implements legal language constructs and that it identifies and rejects illegal language constructs. The testing also identifies behavior that is implementation dependent but permitted by ANSI/MIL-STD-1815A FIPS PUB 119. Six classes of tests are used. These tests are designed to perform checks at compile time, at link time, or during execution.

On-site testing was performed 30 March 1987 through 3 April 1987 in Phoenix, AZ under the auspices of the Software Standards Validation Group, according to Ada Validation Organization policies and procedures. The GCOS-8 V2.0 was hosted on DPS-90 operating under SR3000 V.30002 and SR2500 V.25003

The results of validation are summarized in the following table:

RESULT	TEST CLASS						TOTAL
	A	B	C	D	E	L	
Passed	69	867	1190	17	13	46	2202
Failed	0	0	0	0	0	0	0
Inapplicable	0	0	178	0	0	0	178
Withdrawn	0	7	12	0	0	0	19
TOTAL	69	874	1380	17	13	46	2399

*Ada is a registered trademark of the United States Government (Ada Joint Program Office).

There were 19 withdrawn tests in ACVC Version 1.8 at the time of this validation attempt. A list of these test appears in Appendix D.

Some tests demonstrate that some language features are or are not supported by an implementation. For this implementation, the test determined the following.

- SHORT_INTEGER is not supported.
- LONG_INTEGER is supported.
- SHORT_FLOAT is not supported.
- LONG_FLOAT is supported.
- The additional predefined types LONG_FLOAT, LONG_INTEGER, INTEGER, and FLOAT are supported.
- Representation specifications for noncontiguous enumeration representations are supported.
- The 'SIZE clause is supported.
- The 'STORAGE_SIZE clause is supported.
- The 'SMALL clause is supported.
- Generic unit specifications and bodies can be compiled in separate compilations.
- Pragma INLINE is supported for procedures.
  Pragma INLINE is supported for functions.
- The package SYSTEM is used by package TEXT_IO.
- Mode IN_FILE is supported for SEQUENTIAL_IO.
- Mode OUT_FILE is supported for SEQUENTIAL_IO.
- Instantiation of the package SEQUENTIAL_IO with unconstrained array types is supported.
- Instantiation of the package SEQUENTIAL_IO with unconstrained record types with discriminants is supported.

- Dynamic creation and resetting of files is supported for SEQUENTIAL_IO.
- RESET and DELETE are supported for SEQUENTIAL_ and DIRECT_IO.
- Modes IN_FILE, INOUT_FILE, and OUT_FILE are supported for DIRECT_IO.
- Dynamic creation and resetting of files is supported for DIRECT_IO.
- Instantiation of package DIRECT_IO with unconstrained array types and unconstrained types with discriminants is supported.
- Dynamic creation and deletion of files are supported.
- More than one internal file can be associated with the same external file only for reading.
- An external file associated with more than one internal file can be reset.
- Illegal file names cannot exist.

ACVC Version 1.8 was taken on-site via magnetic tape to Phoenix, AZ. All tests, except the withdrawn tests and any executable tests that make use of a floating point precision greater than SYSTEM.MAX_DIGITS, were compiled on a DPS 90. Class A, C, D, and E tests were executed on a DPS 90.

On completion of testing, execution results for Class A, C, D, or E tests were examined. Compilation results for Class B were analyzed for correct diagnosis of syntax and semantic errors. Compilation and link editing results of Class L tests were analyzed for correct detection or errors.

The Software Standards Validation Group identified 2229 of the 2399 tests in Version 1.8 of the ACVC as potentially applicable to the validation of GCOS-8 V2.0. Excluded were 5 tests with source lines that were too long; 146 tests requiring floating point precision greater than that supported by the implementation; and the 19 withdrawn tests. After the 2229 tests were processed, 27 tests were determined to be inapplicable. The remaining 2202 tests were passed by the compiler.

The Software Standards Validation Group concludes that these results demonstrate acceptable conformance to ANSI/MIL-STD-1815A FIPS PUB 119.

TABLE OF CONTENTS

CHAPTER 1

INTRODUCTION

This Validation Summary Report describes the extent to which a specific Ada compiler conforms to ANSI/MIL-STD-1815A FIPS PUB 119. This report explains all technical terms used within it and thoroughly reports the results of testing this compiler using the Ada Compiler Validation Capability (ACVC). An Ada compiler must be implemented according to the Ada Standard (ANSI/MIL-STD-1815A FIPS PUB 119). Any implementation-dependent features must conform to the requirements of the Ada Standard. The entire Ada Standard must be implemented, and nothing can be implemented that is not in the Standard.

Even though all validated Ada compilers conform to ANSI/MIL-STD-1815A, it must be understood that some differences do exist between implementations. The Ada Standard permits some implementation dependencies--for example, the maximum length of identifiers or the maximum values of integer types. Other differences between compilers result from limitations imposed on a compiler by the operating systems and by the hardware. All of the dependencies demonstrated during the process of testing this compiler are given in the report.

Validation Summary Reports are written according to a standardized format. The report for several different compilers may, therefore, be easily compared. The information in this report is derived from the test results produced during validation testing. Additional testing information is given in section 3.7 and states problems and details which are unique for a specific compiler. The format of a validation report limits variance between reports, enhances readability of the report, and minimizes the delay between the completion of validation testing and the publication of the report.

1.1 PURPOSE OF THIS VALIDATION SUMMARY REPORT

The Validation Summary Report documents the results of the validation testing performed on an Ada compiler. Testing was carried out for the following purposes:

INTRODUCTION

- To attempt to identify any language constructs supported by the compiler that do not conform to the Ada Standard
- To attempt to identify any unsupported language constructs required by the Ada Standard
- To determine that the implementation-dependent behavior is allowed by the Ada Standard

Testing of this compiler was conducted under the supervision of the Software Standards Validation Group according to policies and procedures established by the Ada Validation Organization (AVO). Testing was conducted from 30 March 1987 through 3 April 1987 at Phoenix, AZ.

1.2 USE OF THIS VALIDATION SUMMARY REPORT

Consistent with the national laws of the originating country, the Ada Validation organization may make full and free public disclosure of this report. In the United States, this is provided in accordance with the "Freedom of Information Act" (5 U.S.C. #552). The results of this validation apply only to the computers, operating systems, and compiler versions identified in this report.

The organizations represented on the signature page of this report do not represent or warrant that all statements set forth in this report are accurate and complete, or that the subject compiler has no nonconformances to ANSI/MIL-STD-1815A FIPS PUB 119 other than those presented. Copies of this report are available to the public from:

Ada Information Clearinghouse
Ada Joint Program Office
OUSDRE
The Pentagon, Rm 3D-139
1211 S. Fern, C-107
Washington, DC 20301-3081

or from the Ada Validation Facility (AVF) listed below.

Questions regarding this report or the validation tests should be directed to:

Ada Validation Organization
Institute for Defense Analyses
1801 North Beauregard
Alexandria VA 22311

or to:

Ada Validation Facility
Software Standards Validation Group
Institute for Computer Sciences and Technology
National Bureau of Standards
Building 225, Room A266
Gaithersburg, MD 20899

1.3 RELATED DOCUMENTS

1. Reference Manual for the Ada Programming Language, ANSI/MIL-STD-1815A, FEB 1983.

2. Ada Validation Organization: Policies and Procedures, MITRE Corporation, JUN 1982, PB 83-110601.

3. Ada Compiler Validation Capability Implementers' Guide, SofTech, Inc., DEC 1984.

1.4 DEFINITION OF TERMS

ACVC — The Ada Compiler Validation Capability. A set of programs that evaluates the conformance of a compiler to the Ada language specification, ANSI/MIL-STD-1815A FIPS PUB 119.

Ada Standard — ANSI/MIL-STD-1815A FIPS PUB 119, February 1983.

Applicant — The agency requesting validation.

AVF — Ada Validation Facility. The Federal Software Management Support Center. In the context of this report, the AVF is responsible for conducting compiler validations according to established policies and procedures.

AVO — The Ada Validation Organization. In the content of this report, the AVO is responsible for setting policies and procedures for compiler validations.

Compiler — A processor for the Ada language. In the context of this report, a compiler is any language processor, including cross-compilers, translators, and interpreters.

Failed test — A test for which the compiler generates a result that demonstrates nonconformance to the Ada Standard.

Host — The computer on which the compiler resides.

Inapplicable	A test that uses features of the language that a test compiler is not required to support or may legitimately support in a way other than the one expected by the test.
Passed test	A test for which a compiler generates the expected result.
Target	The computer for which a compiler generates code.
Test	A program that evaluates the conformance of a compiler to a language specification. In the context of this report, the term is used to designate a single ACVC test. The text of a program may be the text of one or more compilations
Withdrawn test	A test which has been found to be inaccurate in checking conformance to the Ada language specification. A withdrawn test has an invalid test objective, fails to meet its test objective, or contains illegal or erroneous use of the language.

1.5 ACVC TEST CLASSES

Conformance to ANSI/MIL-STD-1815A FIPS PUB 119 is measured using the Ada Compiler Validation Capability (ACVC). The ACVC contains both legal and illegal Ada program structured into six test classes: A, B, C, D, E, and L. The first letter of a test name identifies the class to which it belongs. Legal programs are compiled, linked, and executed while illegal programs are only compiled. Special program units are used to report the results of the legal programs.

Class A tests check that legal Ada programs can be successfully compiled and executed. (However, no checks are performed during execution to see if the test objective has been met.) For example, a Class A test checks that reserved words of another language (other than those already reserved in the Ada language) are not treated as reserved words by an Ada compiler. A Class A test is passed if no errors are detected at compile time and the program executes to produce a message indicating that it has passed.

Class B tests check that a compiler detects illegal language usage. Class B tests are not executable. Each test in this class is compiled and the resulting compilation listing is examined to verify that every syntactical or semantic error in the test is detected. A Class B test is passed if every illegal construct that it contains is detected by the compiler.

Class C tests check that legal Ada programs can be correctly compiled and executed. Each Class C test is self-checking and produces a PASSED, FAILED, or NON-APPLICABLE message indicating the result when it is executed.

Class D tests check the compilation and execution capacities of a compiler. Since there are no requirements placed on a compiler by the Ada Standard for some parameters (e.g., the number of identifiers permitted in a compilation, the number of units in a library, and the number of nested loops in a subprogram body), a compiler may refuse to compile a Class D test and still be a conforming compiler. Therefore, if a Class D test fails to compile because the capacity of the compiler is exceeded, the test is classified as inapplicable. If a Class D test compiles successfully, it is self-checking and produces a PASSED or FAILED message during execution.

Each Class E test is self-checking and produces a NOT-APPLICABLE, PASSED or FAILED message when it is compiled and executed. However, the Ada standard permits an implementation to reject programs containing some features addressed by Class E tests during compilation. Therefore, a Class E test is passed by a compiler if it is compiled successfully and executes to produce a PASSED message, or it is rejected by the compiler for an allowable reason.

Class L tests check that incomplete or illegal Ada programs involving multiple, separately compiled units are detected and not allowed to execute. Class L tests are compiled separately and execution is attempted. A Class L test passes if it is rejected at link time--that is, an attempt to execute the main program must generate an error message before any declarations in the main program or any units referenced by the main program are elaborated.

Two library units, the package REPORT and the procedure CHECK_FILE, support the self-checking features of the executable tests. The package REPORT provides the mechanism by which executable tests report results. It also provides a set of identity functions used to detect some compiler optimization strategies and force computations to be made by the target computer instead of by the compiler on the host computer. The procedure CHECK_FILE is used to check the contents of text files written by some of the Class C tests for Chapter 14 of the Ada Standard.

The operation of these units is checked by a set of executable test. These tests produce messages that are examined to verify that the units are operating correctly. If these units are not operating correctly, then the validation is not attempted.

Some of the conventions followed in the ACVC are intended to ensure that the tests are reasonably portable without modification. For example, the tests make use of only the basic set of 55 characters, contain lines with a maximum length of 72 characters, use small numeric values, and place features that may not be supported by all implementations in separate tests. However, some tests contain values that require the test to be customized according to implementation-specific values. The values used for this validation are listed in Appendix C.

A compiler must correctly process each of the tests in the suite and demonstrate conformance to the Ada Standard by either meeting the pass criteria given for the test or by showing that the test is inapplicable to the implementation. Any test that was determined to contain an illegal language construct or an erroneous language construct is withdrawn from the ACVC and therefore, is not used in testing a compiler. The nonconformant tests are given in Appendix D.

CHAPTER 2

CONFIGURATION INFORMATION

2.1 CONFIGURATION TESTED

The candidate compilation system for this validation was tested under the following configuration:

Compiler: GCOS-8 V2.0

Test Suite: Ada Compiler Validation Capability, Version 1.8

Host Computer:

Machine(s):	DPS 90
Operating Systems:	SR2500 V.25003 SR3000 V.30002
Memory Size:	32 mega-words

Target Computer:

Machine(s):	DPS 90
Operating System	SR2500 V.25003 SR3000 V.30002
Memory Size:	32 mega-words

Communications Network:

2.2 CERTIFICATE INFORMATION

Base Configuration:

Compiler: GCOS-8 V2.0

Test Suite: Ada Compiler Validation Capability, Version 1.8

Completion Date: 3 April 1987

Host Computer:

Machine(s): DPS 90

Operating System: SR2500 V.25003
SR3000 V.30002

Target Computer:

Machine(s): DPS 90

Operating System: SR2500 V.25003
SR3000 V.30002

2.3 IMPLEMENTATION CHARACTERISTICS

One of the purposes of validating compilers is to determine the behavior of a compiler in those areas of the Ada Standard that permit implementation to differ. Class D and E tests specifically check for such implementation differences. However, tests in other classes also characterize an implementation. This compiler is characterized by the following interpretations of the Ada Standard:

- Nongraphic characters.

  Nongraphic characters are defined in the ASCII character set but are not permitted in Ada programs, even within character strings. The compiler correctly recognizes these characters as illegal in Ada compilations. The characters are not printed in the output listing. (See test B26005A.)

- Capacities.

  The compiler correctly processes compilations containing loop statements nested to 65 levels, block statements nested to 65 levels, procedures nested to 17 levels. It correctly processes a compilation containing 723 variables in the same declarative part. (See tests D55A03A..H, D56001B, D64005E..G, D29002K)

CONFIGURATION INFORMATION

- Universal integer calculations.

  An implementation is allowed to reject universal integer calculations having values that exceed SYSTEM.MAX INT. This implementation does not reject such calculations and processes them correctly. (See tests D4A002A, D4A002B, D4A004A, and D4A004B.)

- Universal real calculations.

  When rounding to interger is used in a static universal real expression, the value appears to be rounded away from zero. (See test C4A014A.)

- **Predefined types.**

  This implementation supports the additional predefined types LONG_INTEGER, LONG_FLOAT in the package STANDARD. (See test C34001D, C34001G.)

- **Based literals.**

  An implementation is allowed to reject a based literal with a value exceeding SYSTEM.MAX_INT during compilation, or it may raise NUMERIC_ERROR during execution. This implementation raises NUMERIC_ ERROR during execution. (See test E24101A.)

- **Array types.**

  An implementation is allowed to raise NUMERIC_ERROR for an array having a 'LENGTH that exceeds STANDARD.INTEGER'LAST and/or SYSTEM.MAX_INT.

  A packed BOOLEAN array having a 'LENGTH exceeding INTEGER'LAST raises NUMERIC_ERROR when the array objects are declared. (See test C52103X.)

  A packed two-dimensional BOOLEAN array with more than INTEGER'LAST components raises NUMERIC_ERROR when the array type is declared. (See test C52104Y.)

A null array with one dimension of length greater than INTEGER'LAST may raise NUMERIC_ERROR either when declared or assigned. Alternatively, an implementation may accept the declaration. However, lengths must match in array slice assignments. This implementation raises NUMERIC_ERROR when the array type is declared. (See test E52103Y.)

In assigning one-dimensional array types, the entire expression appears to be evaluated before CONSTRAINT_ERROR is raised when checking whether the expression's subtype is compatible with the target's subtype. In assigning two-dimensional array types, the entire expression does not appear to be evaluated before CONSTRAINT_ERROR is raised when checking whether the expression's subtype is compatible with the target's subtype. (See test C52013A.)

- Discriminated types.

  During compilation, an implementation is allowed to either accept or reject an incomplete type with discriminants that is used in an access type definition with a compatible discriminate constraint. This implementation accepts such subtype indications during compilation. (See test E38104A.)

  In assigning record types with discriminants, the entire expression appears to be evaluated before CONSTRAINT_ERROR is raised when checking whether the expression's subtype is compatible with the target's subtype. (See test C52013A.)

- Aggregates.

  In the evaluation of a multi-dimensional aggregate, all choices appear to be evaluated before checking against the index subtype. (See tests C43207A and C43207B.)

In the evaluation of an aggregate containing subaggregates, all choices are evaluated before being checked for identical bounds. (See test E43212B.)

All choices are evaluated before CONSTRAINT_ERROR is raised if a bound in a nonnull range of a nonnull aggregate does not belong to an index subtype. (See test E43211B.)

CONFIGURATION INFORMATION

- Functions.

  The declaration of a parameterless function with the same profile as an enumeration literal in the same immediate scope is rejected by the implementation. (See test E66001D.)

- Representation clauses.

  The Ada Standard does not require an implementation to support representation clauses. If a representation clause is not supported, then the implementation must reject it. While the operation of representation clauses in not checked by Version 1.8 of the ACVC, they are used in testing other language features. Testing indicates that size specifications are supported, that specification of storage for a task activation is supported, and that specification of SMALL for a fixed point type is supported. Enumeration representation clauses including those that specify noncontiguous values appear to be supported. (See tests C55B16A, C87B62A, C87B62B, C87B62C, and BC1002A.)

- Generics.

  When given a separately compiled generic unit specification, some illegal instantiations, and a body, the compiler rejects the body because of the instantiations. (See tests BC3204C and BC3204D.)

- Pragmas.

  The pragma INLINE is supported for procedures. The pragma INLINE is supported for functions. (See tests CA3004E and CA3004F.)

- Input/output.

  The package SEQUENTIAL_IO can be instantiated with unconstrained array types and record types with discriminants.
  The package DIRECT_IO can be instantiated with unconstrained array types and record types with discriminants without defaults. (See tests CE2201D, CE2201E, and CE2401D.)

  More than one internal file can be associated with each external file for SEQUENTIAL_IO for reading only. (See tests CE2107A..F.)

  More than one internal file can be associated with each external file for DIRECT_IO for reading only. (See tests CE2107A..F.)

  An external file associated with more than one internal file can be deleted. (See test CE2110B.)

  More than one internal file can be associated with each external file for text I/O for reading only. (See tests CE3111A..E.)

  Dynamic creation and resetting of a sequential file is allowed. (See test CE2210A.)

  Temporary sequential files are given a name.
  Temporary direct files are given a name.
  Temporary files given names are not deleted when they are closed, but are not accessible after the completion of the main program. (See test CE2108A.)

CHAPTER 3

TEST INFORMATION

3.1 TEST RESULTS

The Software Standards Validation Group identified 2229 of the 2399 tests in Version 1.8 of the Ada Compiler Validation Capability as potentially applicable to the validation of GCOS-8 V2.0. Excluded were 5 tests with source lines that were too long; 146 tests requiring floating point precision greater than that supported by the implementation; and the 19 withdrawn tests. After they were processed 27 tests were determined to be inapplicable. The remaining 2202 tests were passed by the compiler.

The Software Standards Validation Group concludes that the testing results demonstrate acceptable conformance to the Ada Standard.

3.2 SUMMARY OF TEST RESULTS BY CLASS

RESULT	TEST CLASS						TOTAL
	A	B	C	D	E	L	
Passed	69	867	1190	17	13	46	2202
Failed	0	0	0	0	0	0	0
N/A	0	0	178	0	0	0	178
Withdrawn	0	7	12	0	0	0	19
TOTAL	69	874	1380	17	13	46	2399

TEST INFORMATION

3.3 SUMMARY OF TEST RESULTS BY CHAPTER

RESULT

	2	3	4	5	6	7	8	9	10	11	12	14	Total
Passed	99	262	346	246	161	97	139	261	128	32	218	213	2202
Failed	0	0	0	0	0	0	0	0	0	0	0	0	0
N/A	17	63	74	1	0	0	0	1	2	0	0	20	178
W/D	0	5	5	0	0	1	1	2	4	0	1	0	19
TOTAL	116	330	425	247	161	98	140	264	134	32	219	233	2399

3.4 WITHDRAWN TESTS

The following tests have been withdrawn from the ACVC Version 1.8:

C32114A	B37401A	B49006A	C92005A
B33203C	C41404A	B4A010C	C940ACA
C34018A	B45116A	B74101B	CA3005A..D
C35904A	C48008A	C87B50A	BC3204C

See Appendix D for the rationale for withdrawing these tests.

3.5 INAPPLICABLE TESTS

Some tests do not apply to all compilers because they make use of features that a compiler is not required by the Ada Standard to support. Others may depend on the result of another test that is either inapplicable or withdrawn. For this validation attempt, 178 tests were inapplicable for the reasons indicated:

. C96005B - there are no out-of-range values for type DURATION

. CE2401D - Rejected because record sizes greater than (2**18)-1 is not supported.

. - The following are inapplicable because multiple internal files may not access the same external or temporary file.

. CE2107A, CE2107B, CE2107C, CE2107D, CE2107E, CE2110B, CE2111D, CE2111H, CE3111A..CE3111E, CE2114B, CE3115A

. CE2107F - two internal files may not be associated with the same external file for reading.

. CE2108A, CE2108C, CE3112A - temporary files have no names.

. C24113I..C24113M - literals exceeded the limit of 126 characters

. C34001F, C35702A - SHORT_FLOAT is not supported

. C34001D, C55B07B - compiled but not executed because SHORT_INTEGER is not supported.

. CA2009C, CA2009F - these tests have generic bodies that are not in the same compliation as the generic specification.

. These tests are inapplicable because SYSTEM.MAX_ DIGITS = 17; these tests require a greater (floating point) precision:

. C24113N..C24113Y - (12 tests)
. C35705N..C35705Y - (12 tests)
. C35706N..C35706Y - (12 tests)
. C35707N..C35707Y - (12 tests)
. C35708N..C35708Y - (12 tests)
. C35802N..C35802Y - (12 tests)
. C45241N..C45241Y - (12 tests)
. C45321N..C45321Y - (12 tests)
. C45421N..C45421Y - (12 tests)
. C45424N..C45424Y - (12 tests)
. C45521N..C45521Z - (13 tests)
. C45621N..C45621Z - (13 tests)

3.6 SPLIT TESTS

If one or more errors do not appear to have been detected in a Class B test because of compiler error recovery, then the test is split into a set of smaller tests that contain the undetected errors. There were 54 split tests required for this implementation. They are has follows:

B22003A B29001A B2A003A B33301A B35101A B37301B
B37302A B51001A B53009A B54A01C B55A01A B61001C
B61001D B61001F B61001H B61001I B61001M B61001S
B61001W B67001A B67001C B67001D B91001A B91002A
B91002B B91002C B91002D B91002E B91002F B91002G
B91002H B91002I B91002J B91002K B91002L B95030A
B95061A B95061F B95061G B95077A B97102A B97103E
B97104G BA1101B0M BA1101B1 BA1101B2 BA1101B3
BA1101B4 BC1014A BC10AEB BC1202A BC1202B BC1202E
BC1202F

3.7 ADDITIONAL TESTING INFORMATION

3.7.1 Prevalidation

Prior to validation, sets of test results for ACVC Version 1.8 produced by GCOS-8 V2.0 were submitted to the Software Standards Validation Group by the applicant for pre-validation review. Analysis of these results demonstrated that the compiler successfully passed all applicable tests.

The specific configurations submitted for the pre-validation review were as follows:

Host		Target	
Processor	Op. Sys.	Processor	Op. Sys.
DPS 90	SR2500 V.25003	DPS 90	SR2500 V.25003

The DPS 90 results were analyzed and found to be in accordance with the Ada standard.

3.7.2 Test Method

A test magnetic tape containing ACVC Version 1.8 was taken on-site by the validation team. This magnetic tape contained all tests applicable to this validation as well as all tests inapplicable to this validation except for any Class C tests that require floating-point precision exceeding the maximum value supported by the implementation. Tests that were withdrawn from ACVC Version 1.8 were not run. Tests that make use of values that are specific to an implementation were customized before being written to the magnetic tape.

The test tape was written in ANSI standard format and was loaded to disk using Honeywell Corp. utility routines.

Once all tests had been loaded to disk, processing was begun using command scripts provided by Honeywell Information Systems

The validation was executed in batch control mode with the files organized by chapter and class to allow the tests to be run independently and in parallel.

The prevalidation results were verified on-site. The various tests results from the prevalidation execution were captured on disk and used to compare against the on-site results using a utility comparing routine.

The following configurations were tested on-site:

Host	Op. Sys.	Target	Op. Sys.
DPS 90	SR2500 V.25003 SR3000 V.30002	DPS 90	SR2500 V.25003 SR3000 V.30002

3.7.3 Test Site

The validation team arrived at Phoenix, AZ on 30 March 1987 and departed after testing was completed on 3 April 1987.

APPENDIX A

COMPLIANCE STATEMENT

Honeywell Information Systems has submitted the following compliance statement concerning the GCOS-8.

Compliance Statement

Base Configuration:

Compiler: GCOS 8 Ada Compiler Version 2.0

Test Suite: Ada Compiler Validation Capability, Version 1.8

Host Computer:

Machine: DPS 90

Operating System: GCOS 8
Version SR2500
Version SR3000

Target Computer:

Machine: DPS 90

Operating System: GCOS 8
Version SR2500
Version SR3000

Honeywell Information Systems, Large Computer Products Division has made no deliberate extensions to the Ada language Standard.

Honeywell Information Systems, Large Computer Products Division agrees to the public disclosure of this report.

Honeywell Information Systems, Large Computer Products Division agrees to continue to comply with the Ada trademark policy, as defined by the Ada Joint Program Office.

J. E. Wilson Date: 1-22-87

Honeywell Information Systems, Large Computer Products Division
J. R. Wilson
Manager, Advanced Compiler Development

APPENDIX B

III. IMPLEMENTATION DEPENDENCIES
-LRM APPENDIX "F"

TABLE OF CONTENTS

III. IMPLEMENTATION DEPENDENCIES
-LRM APPENDIX "F"

1. IMPLEMENTATION-DEPENDENT PRAGMAS

No implementation dependent pragmas are supported by the GCOS-8 Ada Compiler.

2. IMPLEMENTATION-DEPENDENT ATTRIBUTES

No implementation dependent attributes are supported by the GCOS-8 Ada Compiler.

3. PACKAGE SYSTEM

The specification for package SYSTEM:

```
 Package SYSTEM is

 Type ADDRESS is access INTEGER;
 Type NAME is (DPS8, DPS88, DPS90);
 SYSTEM_NAME : Constant NAME := DPS8;

 STORAGE_UNIT : Constant := 36;
 MEMORY_SIZE : Constant := 256*1024;

 MIN_INT := -2_361_183_241_434_822_606_848;
 MAX_INT := 2_361_183_241_434_822_606_847;
 MAX_DIGITS := 17;
 MAX_MANTISSA := 71;
 FINE_DELTA := 2.0 ** (-70);
 TICK := 0.000016;

 Subtype PRIORITY is INTEGER range 1..15;

 end SYSTEM;
```

4. REPRESENTATION CLAUSES

No representation clauses may be given for a derived type. Representation clauses for non-derived types are accepted as follows:

LENGTH CLAUSE

The compiler accepts only a length clause that specifies the number of storage units to be reserved for a collection.

ENUMERATION REPRESENTATION CLAUSE

Enumeration representation clauses may specify representations only in the range of the predefined type INTEGER.

RECORD REPRESENTATION CLAUSE

A component clause is allowed if and only if:

III. IMPLEMENTATION DEPENDENCIES
-LRM APPENDIX "F"

- The component type is a discrete type different from LONG INTEGER.

- The component type is an array type with a discrete element type different from LONG_INTEGER.

No component clause is allowed if the component type is not covered by the above two inclusions. If the record type contains components not covered by a component clause, they are allocated consecutively after the component with the highest "AT" value. Allocation of a record component without a component clause is always aligned on a word storage unit boundary. Holes created by component clauses are not otherwise used by the compiler.

5. IMPLEMENTATION-DEPENDENT NAMES

No implementation dependent names denoting implementation dependent components are supported.

6. ADDRESS CLAUSES

Address Clauses are not supported.

7. UNCHECKED CONVERSION

Unchecked conversion is only allowed between values of the same "size". In this context, the "size" of an array is equal to that of two access values and the "size" of a packed array is equal to that of two access values and one integer value. This is the only restriction imposed on unchecked conversion.

8. INPUT-OUTPUT

8.1 Introduction

This chapter describes the implementation of Ada Input-Output for the Honeywell Ada Compiler for GCOS 8. The implementation supports all requirements of the Ada language and is an effective interface to the GCOS 8 file system.

This chapter covers three topics. Section 8.2 discusses the requirements of Ada Input-Output systems given in the language definition and provides answers to issues not precisely described in the language definition.

Section 8.3 describes the relation between Ada files and GCOS 8 external files.

Section 8.4 details the implementation dependent FORM parameter of the OPEN and CREATE procedures.

The reader should be familiar with the following documents:

[MIL-STD-1815A-1983] Reference Manual for the Ada Programming

III. IMPLEMENTATION DEPENDENCIES
-LRM APPENDIX "F"

Language

[DH19] DPS8 GCOS8 File Management Supervisor

[DH23] GCOS8 I/O Programming

III. IMPLEMENTATION DEPENDENCIES
-LRM APPENDIX "F"

8.1.1 Declaration of Sequential_IO

```
generic

   type ELEMENT_TYPE is private;

package SEQUENTIAL_IO is

   type FILE_TYPE is limited private;
   type FILE_MODE is(IN_FILE, OUT_FILE);

   procedure CREATE(FILE : in out FILE_TYPE;
                    MODE : in     FILE_MODE := OUT_FILE;
                    NAME : in     STRING    := "";
                    FORM : in     STRING    := "");

   procedure OPEN (FILE : in out FILE_TYPE;
                    MODE : in     FILE_MODE;
                    NAME : in     STRING;
                    FORM : in     STRING := "");

   procedure CLOSE(FILE : in out FILE_TYPE);

   procedure DELETE(FILE : in out FILE_TYPE);

   procedure RESET(FILE : in out FILE_TYPE; MODE : in FILE_MODE);

   procedure RESET(FILE : in out FILE_TYPE);

   function MODE  (FILE : in FILE_TYPE) return FILE_MODE;

   function NAME  (FILE : in FILE_TYPE) return STRING;

   function FORM  (FILE : in FILE_TYPE) return STRING;

   function IS_OPEN(FILE : in FILE_TYPE) return BOOLEAN;

   procedure READ (FILE : in FILE_TYPE; ITEM : out ELEMENT_TYPE);

   procedure WRITE(FILE : in FILE_TYPE; ITEM : in  ELEMENT_TYPE);

   function END_OF_FILE(FILE : in  FILE_TYPE) return BOOLEAN;

private
   type FILE_TYPE is new basic_io.file_type; end SEQUENTIAL_IO;
```

8.1.2 Declaration of Direct_IO

```
generic

   type ELEMENT_TYPE is private;

package DIRECT_IO is

   type FILE_TYPE is limited private;
   type FILE_MODE is(IN_FILE, INOUT_FILE, OUT_FILE);
   type COUNT is range 0..INTEGER'LAST;
   subtype POSITIVE_COUNT is COUNT range 1..COUNT'LAST;

   procedure CREATE(FILE : in out FILE_TYPE;
                    MODE : in     FILE_MODE  := INOUT_FILE;
                    NAME : in     STRING     := "";
                    FORM : in     STRING     := "");

   procedure OPEN (FILE : in out FILE_TYPE;
                    MODE : in     FILE_MODE;
                    NAME : in     STRING;
                    FORM : in     STRING    := "");

   procedure CLOSE(FILE : in out FILE_TYPE);

   procedure DELETE(FILE : in out FILE_TYPE);

   procedure RESET(FILE : in out FILE_TYPE; MODE : in FILE_MODE);

   procedure RESET(FILE : in out FILE_TYPE);

   function MODE  (FILE : in FILE_TYPE) return FILE_MODE;

   function NAME  (FILE : in FILE_TYPE) return STRING;

   function FORM  (FILE : in FILE_TYPE) return STRING;

   function IS_OPEN(FILE : in FILE_TYPE) return BOOLEAN;

   procedure READ (FILE : in FILE_TYPE; ITEM : out ELEMENT_TYPE;
FROM : in POSITIVE_COUNT);
   procedure READ (FILE : in FILE_TYPE; ITEM : out ELEMENT_TYPE);

   procedure WRITE(FILE : in FILE_TYPE; ITEM : in  ELEMENT_TYPE;
TO : in POSITIVE_COUNT);
   procedure WRITE(FILE : in FILE_TYPE; ITEM : in  ELEMENT_TYPE);

   procedure SET_INDEX(FILE : in FILE_TYPE; TO : in POSITIVE_COUNT);

   function INDEX(FILE : in FILE_TYPE) return POSITIVE_COUNT;

   function SIZE(FILE : in FILE_TYPE) return COUNT;
```

III. IMPLEMENTATION DEPENDENCIES
-LRM APPENDIX "F"

```
   function END_OF_FILE(FILE : in  FILE_TYPE) return BOOLEAN;

private
  type FILE_TYPE is new basic_io.file_type; end DIRECT_IO;
```

8.1.3 Declaration of Text_IO

```
package TEXT_IO is

   type FILE_TYPE is limited private;
   type FILE_MODE is(IN_FILE, OUT_FILE);
   type    COUNT is range 0 .. integer'last;
   subtype POSITIVE_COUNT is COUNT range 1 .. COUNT'LAST;
   UNBOUNDED: constant COUNT:= 0; -- line and page length
   subtype FIELD          is INTEGER range 0 .. 75;
         -- max. size of an integer output field
         -- 2#....#
   subtype NUMBER_BASE    is INTEGER range 2 .. 16;
   type TYPE_SET is(LOWER_CASE, UPPER_CASE);

   -- File Management

     procedure CREATE(FILE  : in out FILE_TYPE;
                      MODE  : in     FILE_MODE := OUT_FILE;
                      NAME  : in     STRING    := "";
                      FORM  : in     STRING    := ""
                     );

     procedure OPEN(FILE  : in out FILE_TYPE;
                    MODE  : in     FILE_MODE;
                    NAME  : in     STRING;
                    FORM  : in     STRING    := ""
                   );

   procedure CLOSE(FILE  : in out FILE_TYPE);
   procedure DELETE(FILE  : in out FILE_TYPE);
   procedure RESET(FILE  : in out FILE_TYPE; MODE  : in FILE_MODE);
   procedure RESET(FILE  : in out FILE_TYPE);

   function  MODE(FILE  : in FILE_TYPE) return FILE_MODE;
   function  NAME(FILE  : in FILE_TYPE) return STRING;
   function  FORM(FILE  : in FILE_TYPE) return STRING;

   function  IS_OPEN(FILE  : in FILE_TYPE) return BOOLEAN;

   -- Control of default input and output files

   procedure  SET_INPUT(FILE : in FILE_TYPE);
   procedure  SET_OUTPUT(FILE : in FILE_TYPE);

   function   STANDARD_INPUT   return FILE_TYPE;
   function   STANDARD_OUTPUT  return FILE_TYPE;

   function   CURRENT_INPUT    return FILE_TYPE;
   function   CURRENT_OUTPUT   return FILE_TYPE;

   -- specification of line and page lengths
```

III. IMPLEMENTATION DEPENDENCIES
-LRM APPENDIX "F"

```
   procedure SET_LINE_LENGTH(FILE : in FILE_TYPE; TO : in COUNT);
   procedure SET_LINE_LENGTH(TO : in COUNT);

   procedure SET_PAGE_LENGTH(FILE : in FILE_TYPE; TO : in COUNT);
   procedure SET_PAGE_LENGTH(                     TO : in COUNT);

   function  LINE_LENGTH(FILE : in FILE_TYPE) return COUNT;
   function  LINE_LENGTH                              return COUNT;

   function  PAGE_LENGTH(FILE : in FILE_TYPE) return COUNT;
   function  PAGE_LENGTH                              return COUNT;

   -- Column, Line, and Page Control

     procedure NEW_LINE(FILE : in FILE_TYPE; SPACING : in
POSITIVE_COUNT := 1);
     procedure NEW_LINE(                     SPACING : in
POSITIVE_COUNT := 1);

     procedure SKIP_LINE(FILE : in FILE_TYPE; SPACING : in
POSITIVE_COUNT := 1);
     procedure SKIP_LINE(                     SPACING : in
POSITIVE_COUNT := 1);

     function  END_OF_LINE(FILE : in FILE_TYPE) return BOOLEAN;
     function  END_OF_LINE                      return BOOLEAN;

     procedure NEW_PAGE(FILE : in FILE_TYPE);
     procedure NEW_PAGE                     ;

     procedure SKIP_PAGE(FILE : in FILE_TYPE);
     procedure SKIP_PAGE                     ;

     function  END_OF_PAGE(FILE : in FILE_TYPE) return BOOLEAN;
     function  END_OF_PAGE                      return BOOLEAN;

     function  END_OF_FILE(FILE : in FILE_TYPE) return BOOLEAN;
     function  END_OF_FILE                      return BOOLEAN;

     procedure SET_COL(FILE : in FILE_TYPE; TO : in POSITIVE_COUNT);
     procedure SET_COL(                     TO : in POSITIVE_COUNT);

     procedure SET_LINE(FILE : in FILE_TYPE; TO : in POSITIVE_COUNT);
     procedure SET_LINE(                     TO : in POSITIVE_COUNT);

     function  COL(FILE : in FILE_TYPE) return POSITIVE_COUNT;
     function  COL                      return POSITIVE_COUNT;

     function  LINE(FILE : in FILE_TYPE) return POSITIVE_COUNT;
     function  LINE                      return POSITIVE_COUNT;

     function  PAGE(FILE : in FILE_TYPE) return POSITIVE_COUNT;
```

```
    function  PAGE                                    return POSITIVE_COUNT;

   -- Character Input-Output

   procedure GET(FILE : in FILE_TYPE; ITEM : out CHARACTER);
   procedure GET(                     ITEM : out CHARACTER);
   procedure PUT(FILE : in FILE_TYPE; ITEM :  in CHARACTER);
   procedure PUT(                     ITEM :  in CHARACTER);

   -- String Input-Output

   procedure GET(FILE : in FILE_TYPE; ITEM : out STRING);
   procedure GET(                     ITEM : out STRING);
   procedure PUT(FILE : in FILE_TYPE; ITEM :  in STRING);
   procedure PUT(                     ITEM :  in STRING);

   procedure GET_LINE(FILE : in FILE_TYPE; ITEM : out STRING;
LAST : out NATURAL);
   procedure GET_LINE(                     ITEM : out STRING;
LAST : out NATURAL);
   procedure PUT_LINE(FILE : in FILE_TYPE; ITEM :  in STRING);
   procedure PUT_LINE(                     ITEM :  in STRING);


   -- Generic Package for Input-Output of Integer Types

  generic
      type NUM is range <>;
  package INTEGER_IO is


     procedure GET(FILE  :  in FILE_TYPE; ITEM : out NUM;
WIDTH : in FIELD := 0);
     procedure GET(                       ITEM : out NUM;
WIDTH : in FIELD := 0);

     procedure PUT(FILE  :  in FILE_TYPE;
                   ITEM  :  in NUM;
                   WIDTH :  in FIELD := 10;
                   BASE  :  in NUMBER_BASE := 10);

     DEFAULT_WIDTH : FIELD       := NUM'WIDTH;
     DEFAULT_BASE  : NUMBER_BASE :=       10;

     procedure PUT(ITEM  :  in NUM;
                   WIDTH :  in FIELD := DEFAULT_WIDTH;
                   BASE  :  in NUMBER_BASE := DEFAULT_BASE);

     procedure GET(FROM  : in STRING; ITEM : out NUM; LAST : out POSITIVE);
     procedure PUT(TO    : out STRING;
                   ITEM  :  in NUM;
                   BASE  :  in NUMBER_BASE := DEFAULT_BASE);
```

```
  end INTEGER_IO;

   -- Generic Packages for Input-Output of Real Types

generic
   type NUM is digits <>; package FLOAT_IO is


   procedure GET(FILE : in FILE_TYPE; ITEM : out NUM; WIDTH
: in FIELD := 0);
   procedure GET(                     ITEM : out NUM; WIDTH
: in FIELD := 0);

   DEFAULT_FORE : FIELD :=               2;
   DEFAULT_AFT  : FIELD := NUM'DIGITS - 1;
   DEFAULT_EXP  : FIELD :=               3;

   procedure PUT(FILE : in FILE_TYPE;
                 ITEM : in NUM;
                 FORE : in FIELD := DEFAULT_FORE;
                 AFT  : in FIELD := DEFAULT_AFT;
                 EXP  : in FIELD := DEFAULT_EXP);

   procedure PUT(ITEM : in NUM;
                 FORE : in FIELD := DEFAULT_FORE;
                 AFT  : in FIELD := DEFAULT_AFT;
                 EXP  : in FIELD := DEFAULT_EXP);

   procedure GET(FROM : in STRING; ITEM : out NUM; LAST : out POSITIVE);
   procedure PUT(TO   : out STRING;
                 ITEM :  in NUM;
                 AFT  :  in FIELD := DEFAULT_AFT;
                 EXP  :  in FIELD := DEFAULT_EXP);

end FLOAT_IO;

generic
   type NUM is delta <>; package FIXED_IO is


   procedure GET(FILE : in FILE_TYPE; ITEM : out NUM; WIDTH
: in FIELD := 0);
   procedure GET(                     ITEM : out NUM; WIDTH
: in FIELD := 0);

   DEFAULT_FORE : FIELD := NUM'FORE;
   DEFAULT_AFT  : FIELD := NUM'AFT;
   DEFAULT_EXP  : FIELD := 0;

   procedure PUT(FILE : in FILE_TYPE;
                 ITEM : in NUM;
                 FORE : in FIELD := DEFAULT_FORE;
```

```
                AFT  : in FIELD := DEFAULT_AFT;
                EXP  : in FIELD := DEFAULT_EXP);

   procedure PUT(ITEM : in NUM;
                 FORE : in FIELD := DEFAULT_FORE;
                 AFT  : in FIELD := DEFAULT_AFT;
                 EXP  : in FIELD := DEFAULT_EXP);

   procedure GET(FROM : in STRING; ITEM : out NUM; LAST : out POSITIVE);
   procedure PUT(TO   : out STRING;
                 ITEM :  in NUM;
                 AFT  :  in FIELD := DEFAULT_AFT;
                 EXP  :  in FIELD := DEFAULT_EXP);

end FIXED_IO;

   -- Generic Package for Input-Output of Enumeration Types

generic
   type ENUM is(<>); package ENUMERATION_IO is

   DEFAULT_WIDTH   : FIELD    := 0;
   DEFAULT_SETTING : TYPE_SET := UPPER_CASE;

   procedure GET(FILE : in FILE_TYPE; ITEM : out ENUM);
   procedure GET(                     ITEM : out ENUM);

   procedure PUT(FILE  : in FILE_TYPE;
                 ITEM  : in ENUM;
                 WIDTH : in FIELD      := DEFAULT_WIDTH;
                 SET   : in TYPE_SET   := DEFAULT_SETTING);

   procedure PUT(ITEM  : in ENUM;
                 WIDTH : in FIELD      := DEFAULT_WIDTH;
                 SET   : in TYPE_SET   := DEFAULT_SETTING);

   procedure GET(FROM : in  STRING; ITEM : out ENUM; LAST : out POSITIVE);
   procedure PUT(TO   : out STRING;
                 ITEM : in  ENUM;
                 SET  : in  TYPE_SET := DEFAULT_SETTING);

end ENUMERATION_IO;

private

   type eof_state_type is(eof_not_seen, eof_ahead);

   type file_block is record
                        common           : basic_io.file_type;
                        mode             : file_mode;
                        max_line_length  : natural;
                        max_page_length  : natural;
```

```
                curr_line                : natural;
                curr_page                : natural;
                curr_line_is_end_page    : boolean;
                line_terminator_pending  : boolean;
                line_size_pending        : integer;
                page_terminator_pending  : boolean;
                eof_state                : eof_state_type;
            end record;

    type file_type is access file_block;

end TEXT_IO;
```

III. IMPLEMENTATION DEPENDENCIES
-LRM APPENDIX "F"

8.1.4 Declaration of Low_Level_IO

Low Level input_output is not provided.

III. IMPLEMENTATION DEPENDENCIES
-LRM APPENDIX "F"

8.2 Clarifications of Ada Input-Output Requirements

The Ada Input-Output specification, chapter 14 of [MIL-STD-1815A-1983], provides a number of implementation dependent choices to develop a complete functional specification of the Input-Output packages. These implementation choices are presented below following reference to the appropriate paragraph in the language reference manual [MIL-STD-1815A-1983].

Paragraph	Clarification
14.1(1)	An external file is a GFRC system standard file with media code 6 records (can be terminal directed).
14.1(7)	Named external files will continue to exist at the completion of the main program. Files which have not been closed in an Ada program will be closed by the implementation.
14.1(13)	Two internal Ada files may not be connected to the same GCOS external file.
14.2.1(3)	The name parameter, when non-null, must be a valid GCOS pathname. A permanent file will be created as specified by the pathname. When the name parameter is null, temporary file space will be allocated for the Ada file.
14.2.1(13)	Deletion of a file can occur only when the USERID of the Ada job is either the originator of the file or has MODIFY permissions to that file.
14.2.1(15)	For a sequential or text file, a RESET operation to OUT_FILE mode empties the file of elements.
14.6	The package LOW_LEVEL_IO is empty.

8.3 Basic File Mapping

The relation between Ada files and GCOS 8 files is discussed in this section. The default external characteristics can be modified by the FORM parameter of the OPEN and CREATE operations as discussed in Section 8.4.

8.3.1 Sequential_IO

An Ada sequential file is mapped to a GFRC System Standard sequential file. An element, which cannot be greater than 2**18-1 bytes, is mapped to a control interval on the external file.

8.3.2 Direct_IO

III. IMPLEMENTATION DEPENDENCIES
-LRM APPENDIX "F"

An Ada direct file is mapped to a GFRC System Standard file.

8.3.3 Text_IO

Lines of text are mapped into records on an external file.

For output, the following rules apply.

The Ada line terminators and file terminators are never explicitly stored. Page terminators, except the last, are mapped into a "formfeed" character trailing the last line of the page. (In particular, an empty page (except the last) is mapped into a single record containing only a "formfeed" character). The last page terminator in a file is never represented in the external file. It is not possible to write records containing more than 512 characters. That is, the maximum line length is 511 or 512, depending on whether a page terminator ("formfeed" character) must be written or not. Input of more than 512 characters in a record will raise a USE_ERROR exception.

On input, a "formfeed" trailing a record indicates that the record contains the last line of a page and that at least one more page exists. The physical end of file indicates the end of the last page.

Text-IO files are represented externally as GFRC files in system standard format containing variable length ASCII records (media code 6).

Text_IO standard-input is named "SYSIN" and is read from file code "I*". STANDARD_OUTPUT is named "SYSPRINT" and is written to file code "P*". Both are directed to a terminal when the Ada program is run with program switch word bit 19 set to indicate an interactive program. The "connect to slave" name is the SNUMB of the Ada program.

8.4 FORM Parameter

The FORM string parameter supplied with an OPEN or CREATE procedure enables control over external file properties. The current implementation makes no semantic checks on the content of the FORM string and so does not prohibit illogical settings of external properties of files.

The syntax of the FORM parameter is as follows:

```
form_parameter ::= {space}[param{space param}]
param          ::= - keyword space value
keyword        ::= letter {letter}
letter         ::= A|B|...|Z|a|b|...|z
value          ::= number|letter{letter}
number         ::= 0|1|...|9
```

III. IMPLEMENTATION DEPENDENCIES
-LRM APPENDIX "F"

Notes:

- Letters may be in upper or lower case
- Numbers must be positive decimal integers
- Keywords and values are derived directly from the "$ FFILE" parameters listed in the General Loader Manual (DD10)

9. PACKAGE STANDARD

The implementation dependent predefined types defined in package STANDARD are described in the GCOS 8 Ada User Manual (order number DY76) on pages 4-2 thru 4-4. Copies of this information are inserted here.

10. FILE NAMES

File names follow the conventions and restrictions of the GCOS 8 operating system with the following qualifications. A file name and its superordinate catalog names are separated by slashes ("/"). Names consist of 1 to 12 characters composed of letters, digits, periods, underscores and hyphens. A maximum limit of 10 levels of qualification is allowed. A file name alone signifies that the file is directly below the current user master catalog. Use of passwords is not supported.

PREDEFINED TYPES

This subsection discusses the implementation-dependent predefined types declared in the predefined package STANDARD (Ada Reference Manual, Appendix C) and the relevant attributes of these types.

- BOOLEAN types are implemented as described in the Ada Reference Manual.
- Type CHARACTER is represented in one word.
- Type STRING is represented as one character per byte. Any string comparisons use unsigned compare instructions.

Integer Types

Two predefined integer types are implemented: INTEGER, and LONG_INTEGER, in addition to the anonymous predefined type UNIVERSAL_INTEGER.

They have the following attributes:

```
INTEGER'FIRST       = -34_359_738_368
INTEGER'LAST        =  34_359_738_367
INTEGER'SIZE        =  36

LONG_INTEGER'FIRST  = -2_361_183_241_434_822_606_848
LONG_INTEGER'LAST   =  2_361_183_241_434_822_606_847
LONG_INTEGER'SIZE   =  72
```

Variables of type INTEGER are represented by a single word. Operations on INTEGER variables use 36-bit two's complement arithmetic.

Type LONG_INTEGER is represented by a double word value. Operations on LONG INTEGER use 72-bit two's complement arithmetic.

Floating-Point Types

Two predefined floating-point types are implemented, FLOAT and LONG_FLOAT, in addition to the anonymous predefined type UNIVERSAL_REAL.

- Type FLOAT is represented by the 36-bit hardware hexadecimal floating-point format.
- Type LONG_FLOAT uses the 72-bit hardware, hexadecimal floating-point format. This is similar to the 36-bit format, except that it has a 63-bit mantissa.

The floating-point types have the following attributes:

```
FLOAT'DIGITS                = 6
FLOAT'FIRST                 = -16#0.1#E128
FLOAT'LAST                  = 16#0.FFFFFFE#E127
FLOAT'MACHINE_EMAX          = 127

FLOAT'MACHINE_EMIN          = -128
FLOAT'MACHINE_MANTISSA      = 6
FLOAT'MACHINE_OVERFLOWS     = TRUE
FLOAT'MACHINE_RADIX         = 16
FLOAT'MACHINE_ROUNDS        = TRUE
FLOAT'SAFE_EMAX             = 508
FLOAT'SAFE_LARGE            = 16#0.FFFFFFE#E127
FLOAT'SAFE_SMALL            = 16#0.1#E-128
FLOAT'SIZE                  = 36

LONG_FLOAT'DIGITS           = 17
LONG_FLOAT'FIRST            = -16#0.1#E128
LONG_FLOAT'LAST             = 16#0.FFFF_FFFF_FFFF_FFFE#E127
LONG_FLOAT'MACHINE_EMAX     = 127
LONG_FLOAT'MACHINE_EMIN     = -128
LONG_FLOAT'MACHINE_MANTISSA = 15
LONG_FLOAT'MACHINE_OVERFLOWS = TRUE
LONG_FLOAT'MACHINE_RADIX    = 16
LONG_FLOAT'MACHINE_ROUNDS   = TRUE
LONG_FLOAT'SAFE_EMAX        = 508
LONG_FLOAT'SAFE_LARGE       = 16#0.FFFF_FFFF_FFFF_FFFE#E127
LONG_FLOAT'SAFE_SMALL       = 16#0.1#E-128
LONG_FLOAT'SIZE             = 72
```

Fixed Point Types

Two kinds of anonymous predefined fixed-point types are implemented: FIXED and LONG_FIXED in addition to the anonymous predefined type UNIVERSAL_FIXED or UNIVERSAL_REAL. Note that FIXED and LONG_FIXED are not defined in package STANDARD, but only used here for reference.

For objects of FIXED types, 36 bits are used for the representation; for LONG FIXED, 72 bits are used.

For each of FIXED and LONG_FIXED there exists a virtual predefined type for each possible value of SMALL (refer to the Ada Reference Manual, subsection 3.5.9). The possible values of SMALL are the powers of two that are representable by a LONG_FLOAT value.

The lower and upper bounds of these types are:

```
Lower bound of FIXED types
 = -34_359_738_368 * SMALL
Upper bound of FIXED types
 =  34_359_738_367 * SMALL
Lower bound of LONG_FIXED types
 = -2_361_183_241_434_822_606_848 * SMALL
Upper bound of LONG_FIXED types
 =  2_361_183_241_434_822_606_847 * SMALL
```

A user-defined fixed point type is represented as that predefined FIXED or LONG FIXED type that has the largest value of SMALL not greater than the user-specified DELTA, and which has the smallest range that includes the user-specified range.

Any fixed point type T has the following attributes:

```
T'MACHINE_OVERFLOWS = TRUE
T'MACHINE_ROUNDS    = FALSE
```

The Type DURATION

Type DURATION is represented as a 72-bit LONG-FIXED representation.

The predefined fixed point type DURATION has the following attributes:

```
DURATION'DELTA   = 0.0000015625
DURATION'FIRST   = -2_251_799_813_685_248.0
DURATION'LAST    = 2_251_799_813_685_247.999999046325683593 75
DURATION'SIZE    = 72
DURATION'SMALL   = 2#1.0#E-20
```

Type COUNT

The range of the type COUNT defined in package DIRECT_IO and in package TEXT_IO is 0..INTEGER'LAST.

APPENDIX C

TEST PARAMETERS

Certain tests in the ACVC make use of implementation-dependent values, such as the maximum length of an input line and invalid file names. A test that makes use of such values is identified by the extension. TST in its file name. Actual values to be substituted are identified by names that begin with a dollar sign. A value is substituted for each of these names before the test is run. The values used for this validation are given below.

Name and Meaning	Value
$BIG_ID1 Identifier of size MAX_IN_LEN with varying last character.	BI<120x"G">_ID1
$BIG_ID2 Identifier of size MAX_IN_LEN with varying last character.	BI<120x"G">_ID2
$BIG_ID3 Identifier of size MAX_IN_LEN with varying last character.	BI<100x"G">3<20x"G">_ID
$BIG_ID4 Identifier of size MAX_IN_LEN with varying last character.	BI<100x"G">4<20x"G">_ID
$BIG_INT_LIN An integer literal of value 298 with enough leading zeroes so that it is MAX_IN_LEN characters long.	<123x"0">298

Name and Meaning	Value
$BIG_REAL_LIT A real literal that can be either of floating or fixed point type, has value 690.0, and has enough leading zeroes to be MAX_IN_LEN characters long.	<120x"0">69.0E1
$BLANKS Blanks of length MAX_IN_LEN - 20	<106x" ">
$CNT_LAST Value of CNT'LAST in TEXT_IO package.	34_359_738_367
$EXTENDED_ASCII_CHARS A string literal containing all the ASCII characters with printable graphics that are not in the basic 55 Ada character set.	abcdefghijklmnopqrstuvwxyz!$%?@[\]^'{}~"
$FIELD_LAST Value of Field'LAST in TEXT_IO package.	75
$FILE_NAME_WITH_BAD_CHARS An illegal external file name that either contains invalid characters or is too long.	F(*FILE
$FILE_NAME_WITH_WILD_CARD_CHAR An external file name that either contains a wild card character or is too long.	N234567890123
$GREATER_THAN_DURATION A universal real value that lies between DURATION'BASE'LAST and DURATION'LAST or any value in the range of DURATION	2_600_000_000_000_000.0
$GREATER_THAN_DURATION_BASE_LAST The universal real value that is greater than DURATION'BASE'LAST.	2_600_000_000_000_000.0
$ILLEGAL_EXTERNAL_FILE_NAME Illegal external file name.	F@LENAME

Name and Meaning	Value
$ILLEGAL_EXTERNAL_FILE_NAME2 Illegal external file names.	MUCHTOOLONGFORFILENAME
$INTEGER_FIRST The universal integer literal expression whose value is INTEGER'FIRST.	-34_359_738_368
$INTEGER_LAST The universal integer literal expression whose value is INTEGER'LAST.	34_359_738_367
$LESS_THAN_DURATION A universal real value that lies between DURATION'BASE'FIRST and DURATION'FIRST or any value in the range of DURATION.	-2_600_000_000_000_000.0
$LESS_THAN_DURATION_BASE_FIRST The universal real value that is less then DURATION'BASE'FIRST.	-2_600_000_000_000_000.0
$MAX_DIGITS Maximum digits supported for floating-point types.	17
$MAX_IN_LEN Maximum input line length permitted by the implementation.	126
$NAME A name of predefined numeric type other than FLOAT, INTEGER, SHORT_FLOAT, SHORT_INTEGER, LONG_FLOAT, or LONG_INTEGER,	
$NEG_BASED_INT A based integer literal whose highest order nonzero bit falls in the sign bit position of the representation for SYSTEM.MAX_INT.	16#FFFFFFFFFFFFFFFF#
$NON_ASCII_CHAR_TYPE An enumerated type definition for a character type whose literals are the identifier NON_NULL and all non-ASCII characters with printable graphics.	(NON_NULL)

APPENDIX D

WITHDRAWN TESTS

Some tests are withdrawn from the ACVC because they do not conform to the Ada Standard. When testing was performed, the following 19 tests had been withdrawn at the time of validation testing for the reasons indicated:

- B4A010C: The object_declaration in line 18 follows a subprogram body of the same declarative part.
- BC3204C: The file BC3204C4 should contain the body for BC3204CO as indicated in line 25 of BC3204C3M.
- C35904A: The elaboration of subtype declarations SFX3 and SFX4 may raise NUMERIC_ERROR (instead of CONSTRAINT_ERROR).
- C41404A: The values of 'LAST and 'LENGTH are incorrect in IF statements from line 74 to the end of the test.
- C48008A: This test requires that the evaluation of default initial values not occur when an exception is raised by an allocator. However, the Language Maintenance Committee (LMC) has ruled that such a requirement is incorrect (AI-00397/01).
- C32114A: An unterminated string literal occurs at line 62.
- B33203C: The reserved word "IS" is misspelled at line 45.
- C34018A: The call of function G at line 114 is ambiguous in the presence of implicit conversions and inconsistent without.
- B37401A: The object declarations at lines 126-135 follow subprogram bodies declared in the same declarative part.
- B45116A: ARRPRIBL1 and ARRPRIBL2 are initialized with a value of the wrong type (PRIBOOL_TYPE instead of ARRPRIBOOL_TYPE) at line 41.
- B49006A: Object declaratives at lines 41 and 50 are terminated incorrectly with colons; "END CASE;" is missing from line 42.

- B74101B: The "BEGIN" at line 9 is mistaken; it causes the declarative part to be treated as a sequence of statements.

- C87B50A: The call of "/=" at line 31 requires a "USE" clause for package A.

- C92005A: At line 40, "/=" for type PACK.BIG_INT is not visible without a "USE" clause for package PACK.

- C940ACA: This test assumes that allocated task TT1 will run prior to the main program, and thus assign SPYNUMB the value checked for by the main program; however, such an execution order is not required by the Ada Standard, so the test is erroneous.

- CA3005A..D (4 tests): No valid elaboration order exists for these tests.

END OF LIST

8704035 1 08037

FSV87VSRHIS518A

Ada* COMPILER
VALIDATION SUMMARY REPORT:
Honeywell Information Systems
GCOS-8 V2.0

Completion of On-Site Validation:
3 April 1987

Prepared By:
Software Standards Validation Group
Institute for Computer Sciences and Technology
National Bureau of Standards
Building 225, Room A266
Gaithersburg, MD 20899

Prepared For:
Ada Joint Program Office
United States Department of Defense
Washington, D.C.

---

*Ada is a registered trademark of the United States Government (Ada Joint Program Office).

END

8-87

DTIC